DESIGN & ANALYSIS OF AN AREA-EFFICIENT BINARY MULTIPLIER ARCHITECTURE

*** Bobby Nelson, ** Dr Viswanath H L**
** PhD Scholar, ** Professor: Dept of ECE,*
CHRIST (Deemed to be University), Faculty of Engineering
Kengeri, Bangalore, India

ABSTRACT

The paper entails the design of an extremely area-efficient binary multiplier chiefly for use in Application Specific Integrated Circuits (ASICs) and embedded systems that rely on minimalistic area-constraints and expenses at the cost of speed. The design of the multiplier core has been based on the Urdhva-Tiryagbhyam (UT) theorem, an ancient simplification technique as described in the Vedic Scripts, for a simplified architecture. The process of multiplication has also been further improved using the Karatsuba algorithm to reduce the overall delay of the design. The final implementation has then been carried out using Gate-Diffusion-Input (GDI) based cells for further improvements in the area-constraints.

Keywords: *Binary Multiplier, Gate-Diffusion-Input, Karatsuba Algorithm, Urdhva-Tiryagbhyam Theorem.*

INTRODUCTION

Multiplication happens to be one of the most elaborate digital processes and offers a good scope for improvement in terms of area, delay as well as power efficiency. The multiplier data-path has extremely complex architecture among all the other arithmetic components and is usually simplified for a fast response in general processors because speed is the most crucial aspect of digital integrated circuit designing in general computing applications. However, for the more specific utilities as in the case of Application Specific Integrated Circuits (ASICs), the crux of the design, primarily happens to be the chip area as it is directly proportional to the manufacturing costs of the IC. The ASICs tend to have a minimal architecture because cost-efficiency is the key to their design. However, the conventional hierarchical array multiplication [11], being a simple looped method in which a multiplicand is repetitively multiplied with the individual bits of second multiplicand and the partial product terms are then finally added to yield the product, requires extensive area for its realization and surpasses the expenditure budget of the overall design by heavy margins.

The scope of the paper is to design a highly area-efficient binary multiplier, which is well suited to the ASIC design constraints, using the combined benefits of the Karatsuba algorithm and the Urdhva-Tiryagbhyam (UT) theorem [9], with the implementation being carried out in Gate-Diffusion-Input (GDI) based cells.

METHODOLOGY

The UT theorem has been used in the design of the core multiplier to reduce the complications in the design and to resolve critical adder delays. The Karatsuba algorithm is effectively used to minimize the area-constraints as well as the delay. The implementation of the design in GDI based cells serves to reduce the area-constraints furthermore. The design methodology has been elaborately described below with respect to each of these aspects.

A. UT Theorem

The term Urdhva-Tiryagbhyam literally translates to "Vertically and Crosswise" from Sanskrit. The UT theorem [4], [14], as defined in the ancient Vedic Sutras, is a tool used for the simplification of the multiplication process. Traditionally, it applies to the decimal number system. However, we apply the same ideas to the binary number system to make the proposed algorithm compatible with the digital hardware. The process of multiplication is broken down into two stages, the first stage being the generation of partial products of varying bit-widths and the second being their addition to yield the final product. The process can be summed up into following expressions mathematically wherein *A* and *B* represent the 9-bit multiplicands and terms *D-T* represent the partial products. The Table 1, represents the addition process for the partial products to yield the final product *'$X_{18\text{-}bits}$'*.

$D = A_0B_0$

$E = A_0B_1+A_1B_0$

$F = A_0B_2+A_1B_1+A_2B_0$

$G = A_0B_3+A_1B_2+A_2B_1+A_3B_0$

$H = A_0B_4+A_1B_3+A_2B_2+A_3B_1+A_4B_0$

$I = A_0B_5+A_1B_4+A_2B_3+A_3B_2+A_4B_1+A_5B_0$

$J = A_0B_6+A_1B_5+A_2B_4+A_3B_3+A_4B_2+A_5B_1+A_6B_0$

$K = A_0B_7+A_1B_6+A_2B_5+A_3B_4+A_4B_3+A_5B_2+A_6B_1+A_7B_0$

$L = A_0B_8+A_1B_7+A_2B_6+A_3B_5+A_4B_4+A_5B_3+A_6B_2+A_7B_1+A_8B_0$

$M = A_1B_8+A_2B_7+A_3B_6+A_4B_5+A_5B_4+A_6B_3+A_7B_2+A_8B_1$

$N = A_2B_8+A_3B_7+A_4B_6+A_5B_5+A_6B_4+A_7B_3+A_8B_2$

$O = A_3B_8+A_4B_7+A_5B_6+A_6B_5+A_7B_4+A_8B_3$

$P = A_4B_8+A_5B_7+A_6B_6+A_7B_5+A_8B_4$

$Q = A_5B_8+A_6B_7+A_7B_6+A_8B_5$

$R = A_6B_8+A_7B_7+A_8B_6$

$S = A_7B_8+A_8B_7$

$T = A_8B_8$

B. Karatsuba Algorithm

The Karatsuba algorithm [12], [13], is a fast multiplication algorithm, developed by Anatoly Karatsuba in 1960, published in 1962 [2]. The algorithm can effectively reduce the multiplication of two n-digit numbers to at most $N^{\log_2 3}$ ($N^{1.59}$) single-digit multiplications. It is therefore faster than the classical algorithm, which requires N^2 single-digit products.

For example, it requires 3^{10} = 59,049 single-digit multiplications, using the Karatsuba algorithm, to multiply two 1024-digit numbers ($n = 1024 = 2^{10}$), whereas the classical algorithm requires $(2^{10})^2$ = 1,048,576 multiplications.

Considering two 16-bit wide binary variables, *A* and *B*, the numbers are broken down as follows into their two significant halves:

$A = A_H * 2^8 + A_L$

$B = B_H * 2^8 + B_L$

Hence,

$AB = (A_H * 2^8 + A_L)*(B_H * 2^8 + B_L)$

The above expression shows the multiplication of the two binary numerals and when expanded they yield four multiplication terms represented as:

$X = A_H B_H$

$Y = A_H B_L + A_L B_H$

$Z = A_L B_L$

$AB = X*2^{16} + Y*2^8 + Z$

So, the long multiplication technique here requires a total of four 8-bit multipliers, with *Y* itself requiring two. However, going by the Karatsuba approach, the multiplication terms shall be reduced as:

$V = (A_H + A_L)*(B_H + B_L) - X - Z = Y$

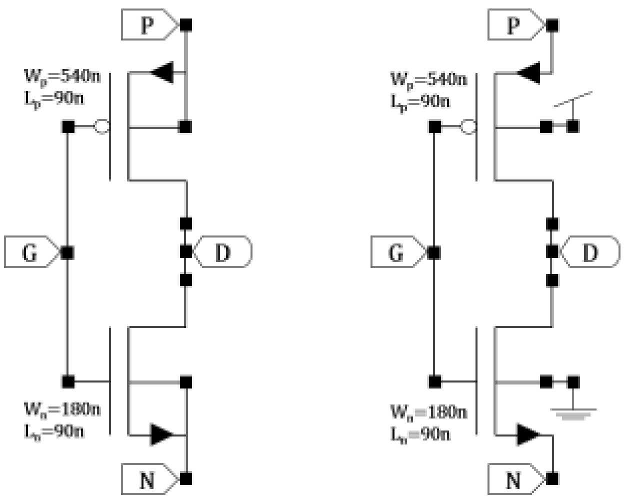
Here, *V* represents an alternate term which does not require just one multiplier and is numerically equivalent to *Y*. Hence, the expression shall be,

$AB = X*2^{16} + V*2^8 + Z$

The above expression depicts the numerical representation of the Karatsuba algorithm involving the calculation of three multiplicands as opposed to the four required as in the case of classical approach. However, the caveat associated with this algorithm is that the term *V* might not necessarily have the same multiplication bit-width as *X* and *Z*. The addition of two 8-bit numbers may yield a 9-bit result at max. Hence, the UT multiplier core needs to have a bit-width of 9-bits.

C. GDI Cells

The GDI design technique [1], [5], was introduced as a promising alternate approach of implementing logical functions minimally in two transistors. GDI methodology allows the implementation of a wide range of complex logic functions using merely two transistors.



(a) (b)

Fig. 1. (a) Basic GDI cell (b) Improved GDI-based cell

Fig. 1. (a), shows the basic construction of a GDI cell and the improved GDI-based cell has been shown in Fig. 1. (b). The cells have been designed with a W_p/W_n ratio of 3. The basic GDI cells suffer from excessive partial swing problems which renders them futile for use in any cascaded connections. The improved GDI-based cell shown in Fig. 1. (b) reduces the swing problems to some extent however the actual implementation of the sub-circuits still requires a few necessary adjustments as shown in the implementation section.

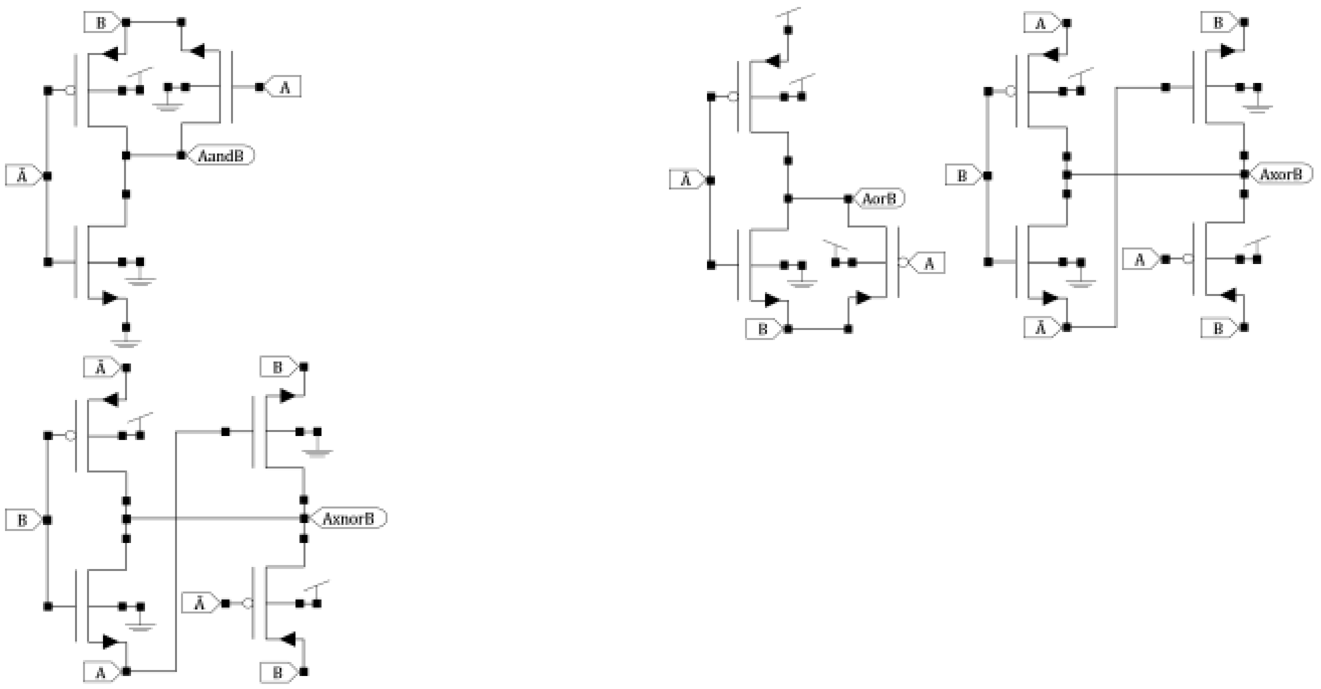
TABLE II

FUNCTION IMPLEMENTATION IN GDI CELLS

N	P	G	D	FUNCTION
B	0	A	A.B	AND
0	1	A	A’	INVERTER
C	B	A	A’B+AC	MUX
1	B	A	A+B	OR
B	B’	A	AB+A’B’	XNOR
B’	B	A	A’B+AB’	XOR

IMPLEMENTATION

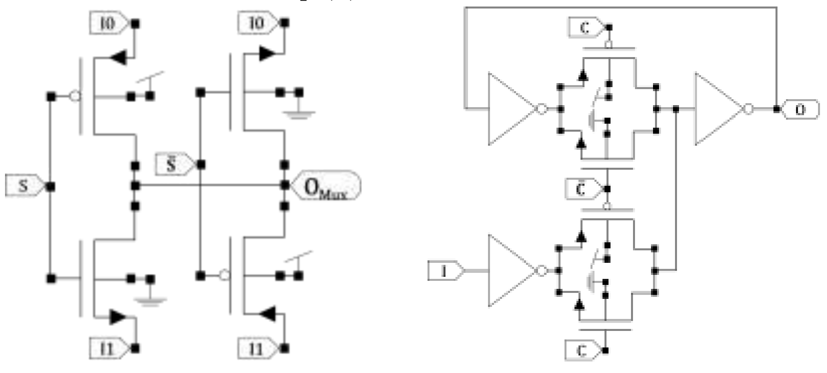
The GDI-based cells [7] are used in the design of the sub-circuit blocks for the proposed multiplier. The cells are all optimized for a full output swing at the cost of a slight increase in the MOSFET-count. The MOSFETs have a uniform dimensional specification all through the design as shown in the Fig. 1. The design of these basic logic gates is shown in Fig. 2. The AND & OR gates need an extra MOSFET to compensate for the partial swing while the XOR & XNOR [7], [8], gates require two each for the same purpose. The gates are also used with inverter stages in the middle that act as buffers.



(a) (b) (c) (d)

Fig. 2. Sub-circuits for the basic logic gates

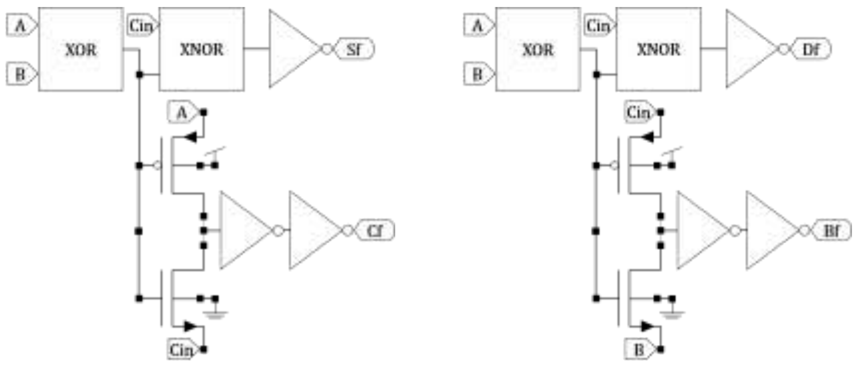
The Fig. 3. (a), depicts the sub-circuit of a 2:1 Multiplexer with a full-swing output. The multiplexer. in the design, serves an important purpose of separating the input multiplicands to be fed into the single 9-bit UT multiplier as per the Eq. (2). The Fig. 3. (b), depicts a sub-circuit for a positive latch [15] based on the Multiplexer logic. The latch is integral in the proposed design as it serves the purpose of using the combinatorial multiplier core, sequentially by storing the individual products of the two significant halves as seen in the Eq. (2) after each iteration.



(a) (b)

Fig. 3. (a) 2:1 Multiplexer (b) Latch

The other crucial sub-circuits under consideration are the extensively used adders and subtractors in the design. From the design of the 9-bit UT Multiplier core to the ripple-carry-adders and subtractors in the overall design, these constitute an integral component in the design of the proposed multiplier and their extensive usage requires them to be as area-efficient as possible. The design of the half-adders and half-subtractors has been carried out rather simplistically using mere XOR & AND sub-circuits which amount to 9 MOSFETs each. However, the design of the full-adder [6], and the full-subtractor sub-circuits has been carried out more minimally as shown in the Fig. 4. The XOR and XNOR sub-circuits do require an input inverter each.



(a) (b)

Fig. 4. (a) Full-adder (b)Full-subtractor

The Fig. 5 presents the design of the 9-bit UT multiplier core. As the name implies, the multiplier is based on the UT theorem discussed earlier and works in two stages, namely the generation of partial products from Eq. (1) and their additions as per the Table 1. The 9-bit multiplier has a dual purpose of multiplying 8-bit as well as 9-bit multiplicands. The first two stages involve the multiplication of the most and the least significant 8-bits. The 9th bit in these stages is simply grounded off via the MUXs. The third stage utilizes the full 9-bit capability of the UT multiplier for the additive multiplicands.

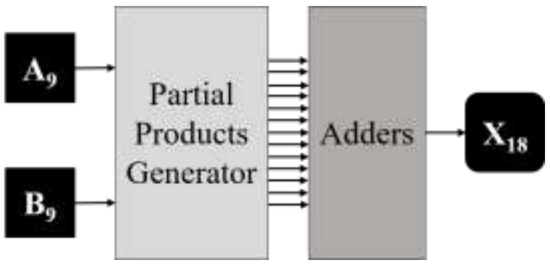


Fig. 5. 9-bit UT Multiplier

The final proposed multiplier design is shown in the Fig. 6. It utilizes the Karatsuba algorithm and is implemented using all the sub-circuits discussed so far. The final multiplier output is obtained with the realization of Eq. (2), through the *Adders & Subtractors* block as *'$P_{32\text{-}bits}$'*. The first control signal *'S_2'* is a 2-bit selection line signal for the MUX to differentiate among the various stages of multiplication. The second control signal *'W_2'* is a 2-bit word-line signal for enabling and disabling the latches used for storing the multiplication results *'X'* and *'Z'* as in the Eq. (2). The generation of these control signals may add as an overhead to the clock unit of the system. The clock scheme used for the control signals has been shown in Fig. 7.

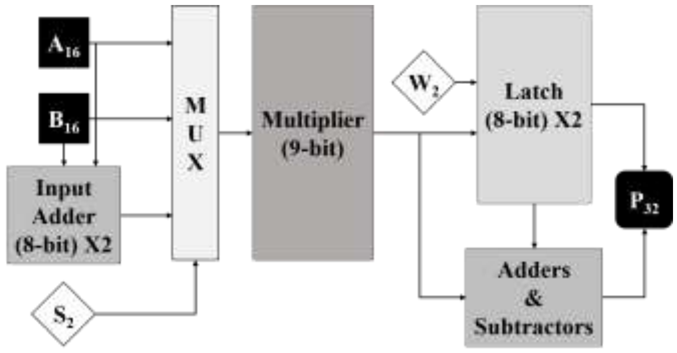


Fig. 6. The proposed 16-bit multiplier

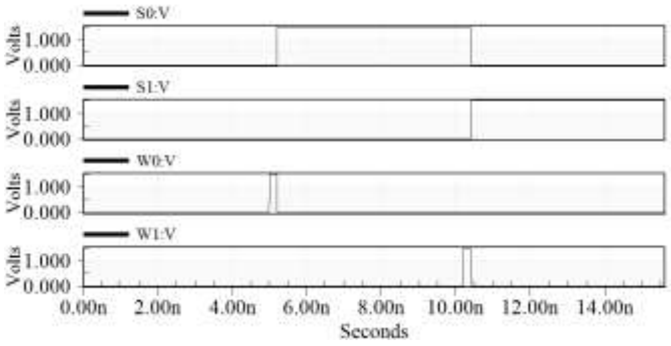


Fig. 7. Clock scheme for the proposed multiplier's control signals

RESULTS & ANALYSIS

The proposed multiplier has been designed and simulated at 90nm technology with input voltages of 1.5 V, and with the uniform dimensional specifications from Fig. 1. The Table 3. presents the analysis of the various designed circuits. The worst-case delay and the average power consumption for each circuit have been listed in the table. The area has been represented by the MOSFET count of each circuit however, the input inverters have not been considered for the logic gates.

TABLE III

ANALYSIS OF DESIGNED CIRCUITS

Circuits	MOSFETs	Delay, ns	Power, µW
INVERTER	2	0.0299	2.473
AND	3	0.0076	0.152
OR	3	0.2256	0.501
MUX	4	0.0375	0.003
XNOR	4	0.0440	0.033
XOR	4	0.0440	0.161
LATCH	10	0.0572	34.90
Half-Adder	12	0.1542	11.44
Half-Subtractor	12	0.1044	15.54
Full-Adder	20	0.2765	19.43
Full-Subtractor	20	0.2768	21.06
UT Multiplier	1842	4.2485	1514
Proposed Multiplier	**3747**	**15.004**	**5376**

The circuits have been arranged in the order of their MOSFET count. The worst-case delay and power-dissipation for the multipliers have been calculated using several pseudorandom input sequences. The Fig. 8. shows the simulation results of the proposed multiplier architecture for an input combination of all 1's. The traces at the high-state, i.e., at 1.5 V, have been highlighted in black.

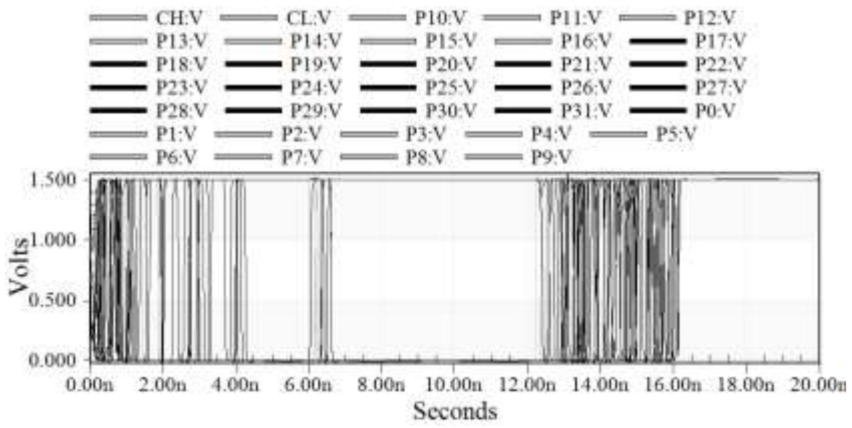


Fig. 8. Simulation of proposed multiplier for an all 1's input combination

The Table. 4 presents a comparative analysis of a few multiplier architectures with the proposed multiplier in terms of the area or the MOSFET count. The reduction in the area has been evident by a noticeably large margin and hence the proposed design makes up for an area-efficient alternate for the ASICs and for other circuits requiring an extremely compact multiplier design.

TABLE IV
COMPARISON BETWEEN MULTIPLIER ARCHITECTURES

Multiplier	**MOSFETs**
CMOS Array Multiplier [11]	16032
CMOS UT Multiplier [3]	12864
Booth-Wallace Tree Multiplier [10]	7858
Proposed Multiplier	3747

CONCLUSION

A new highly area-efficient design for a compact 16-bit multiplier has been proposed and explained in detail. It has been effectively established in Table. 4., that the proposed multiplier uses the lowest chip area in terms of the MOSFET count as compared with the other multiplier architectures and is therefore favorable for use in ASICs. The proposed design, implemented in the 90nm technology using GDI-based cells, has been found to yield conducive results in all the tests carried out upon it.

REFERENCES

[1] A. Morgenshtein, A. Fish, and I. Wagner, “Gate-diffusion-input (GDI): a power-efficient method for digital combinatorial circuits,” *IEEE Trans. on VLSI Syst.*, Vol. 10, No. 5, pp. 566-581, October 2002.

[2] A. Karatsuba, and Y. Ofman, “Multiplication of multidigit numbers on automata,” *Soviet Physics – Doklady,* Vol. 7, No.7, pp. 595-596, January 1963.

[3] A. Somani, D. Jain, S. Jaiswal, K. Verma, and S. Kasht, “Compare vedic multipliers with conventional hierarchical array of array multiplier,” *International J. of Computer Tech. and Electronics Engg.*, Vol. 2, No. 6, pp. 52-55, December 2012.

[4] A. Kareem, M. Vardhana, and P. Kumar, “VLSI implementation of high-speed-low-power-area-efficient multiplier using modified vedic mathematical techniques”, *Recent Patents on Computer Science*, Vol. 9, No. 3, pp. 216-221, January 2017.

[5] S. Kaur, B. Singh, and D.K. Jain, “Design and performance analysis of various adders and multipliers using GDI technique,” *International J. of VLSI Design & Comm. Syst.*, Vol. 6, No. 5, pp. 45-56, October 2015.

[6] A. Shams, and M. Bayoumi, “A novel high-performance CMOS 1-bit full-adder cell,” *IEEE Trans. on Circuits and Syst. II: Analog and Digital Signal Processing*, Vol. 47, No. 5, pp. 478-481, May 2000.

[7] S. Mohan, and N. Rangaswamy, “GDI based full adders for energy efficient arithmetic applications”, *Engg. Sci. and Tech., an International J.*, Vol. 19, pp. 485-496, 2016.

[8] J.M. Wang, S.C. Fang, and W.S. Feng, “New effective designs for XOR and XNOR functions on the transistor level,” *IEEE J. of Solid-State Circuits*, Vol. 29, No. 7, pp. 780-786, July 1994.

[9] S. Arish, and R.K. Sharma, “An efficient binary multiplier design for high speed applications using karatsuba algorithm and urdhva-tiryagbhyam algorithm” in *Proc. GCCT,* Thuckalay, India, 2015, pp. 192-196.

[10] F. Jalil, “M*N booth encoded multiplier generator using optimized wallace trees,” *IEEE Trans. on VLSI Syst.,* Vol. 1, No. 2, pp. 120-125, June 1993.

[11] A. Asati, and Chandrashekhar, “A high-speed, hierarchical 16x16 array multiplier design,” in *Proc. IMPACT* ’09, Aligarh, India, 2009 pp. 161-164.

[12] R.P. Brent, and P. Zimmermann, “Integer Arithmetic,” in *Modern computer arithmetic,* Cambridge, Cambridge University Press, 2010, pp. 3-14.

[13] K.O. Geddes, S.R. Czapor, and G. Labahn, “Arithmetic of Polynomials, Rational Functions, and Power Series,” *Algorithms for computer algebra*, Springer, 2014, pp. 118-119.

[14] A. Nicholas, K. Williams, and J. Pickles, “*Vertically and Crosswise*”, Castle Douglas, Scotland UK: Inspiration Books, 2010.

[15] J.M. Rabaey, A.P. Chandrakasan, and B. Nikolic, “Designing Sequential Logic Circuits,” in *Digital Integrated Circuits – A design perspective,* 2nd ed., India, Pearson Education, 2003, pp. 280-283.